NOISE & VIBRATION Research Group
KU Leuven Department of Mechanical Engineering

KU LEUVEN

Mechanical Engineering · Departement Werktuigkunde

Internal report

KU Leuven in-house AMESim functionalities for state estimation and model predictive control

Identifier: IR_2014_1

Date: 28/04/2014

Author: Jan Croes

Promotor: Wim Desmet

1. Introduction

This library makes it possible to use AMESim in a framework for model predictive control and state estimation. It calls the solver to integrate from one timestep to another and provides the sensitivities with respect to the states and the inputs with respect to the starting values. An additional feature is that the parameters can be changed at will at any point in time. The inputs are treated as parameters by AMESim but the library makes a specific disctinction w.r.t. functionalities.

The functions are provided in a format suitable for ACADO and do not require the use of any additional library. The programming language is C code and the framework is developed with Microsoft Visual Studio 2008 (MSVS). As the source code is based on the GNU libraries 'cblas' and 'GSL', the migration to a UNIX platform remains a possibility.

Special actions are taken to ensure robustness:

- Dealing with discontinuities
- Dealing with bounds on the states, inputs and parameters

This package includes:

- Two header files that allow the user to acces these functions;
- The static library that contains all the compiled code;
- An additional executable that generates a header file with all the necessary data based on the specific model;
- An example script that illustrates all the necessary functions;
- A script that extends the standard possibilities of the AMESim dll interfacing.

2. Description of the functionalities

Function 1

void initialiseModelEvalfFunctions(char *path);

Description

This function loads the dll, which creates the functions necessary to interact with the AMESim model. This is a combination of existing AMESim functions, extended for parameter estimation.

Inputs/outputs

- path: path to the AMESim file

Function 2

```
void initialiseModelEvalfParameters(    int Nsi,
                                        int NoPi,
                                        int NoIi,
                                        int Obsi,
                                        int nrofInterStepsi,
                                        char *path,
                                        char **observations,
                                        char **input_vector,
                                        double *peri);
```

Description

This function creates a list of all the necessary variables that will be used during the simulation.

Inputs/outputs

- Nsi: number of states
- NoPi: number of parameters
- NoIi: number of inputs
- Obsi: number of variables one want to observe
- nrofInterStepsi:number of intermediate steps within a timestep to deal with discontinuities
- path: path to the AMESim file
- observations: an array of strings that says which variables one wants to observe
- input_vector: an array of strings of the states, inputs and parameters one wants to change

Function 3

```
void setAparameter(     double Aparam,
                        int Nr);
```

Description

This function allows to set a parameter at any point in time.

Inputs/outputs

- Aparam: value for the parameter
- Nr: number of parameter starting from zero

Function 4

```
void setConstraints(    double *min,
                        double *max,
                        int *stateNrs,
                        int NoCSi);
```

Description

This function sets constraints on some of the states and all of the inputs and parameters.

Inputs/outputs

- min: array of minimum values for the states,inputs and parameters
- max: array of maximum values for the states,inputs and parameters
- stateNrs: array that indicates which of the (augmented) states are constraint
- NoCSi: number of constraint states

Function 5

```
void evaluatey(         double t,
                        double *x,
                        double *u,
                        double *y);
```

Description

This function evaluates some of the variables for a particular set of states.

Inputs/outputs

- x: set of states
- u: set of inputs
- t: the starting time of the simulation
- x: set of variables

Function 6

```
void setTheSolver(          double dt,
                            int NofIPoints);
```

Description

This function sets the solver specifics.

Inputs/outputs

- dt: timestep of your MPC algorithm
- NofIPoints: number of points that needs to be printed in the AMESim results file within 1 timestep

Function 7

```
void closeModelEvalf();
```

Description

This function finishes up the simulation, unloads the dll and clears up the memory.

Function

```
void integrate(         double *x,
                        double *Adouble,
                        double *Bdouble,
                        double *u,
                        double t);
```

Description

This function integrates over a particular timestep and gives the sensitiveties with respect to the initial state and input.

Inputs/outputs

- x: set of states
- A: sensitivity w.r.t. the states
- B: sensitivity w.r.t. the inputs
- u: set of inputs
- t: the starting time of the simulation

Function 8

```
void CDevalf(   double *x,
                double *u,
                double *Cdouble,
                double *Ddouble,
                double t);
```

Description

This function provides the sensitivities with respect to the states for a set of variables.

Inputs/outputs

- x: set of states
- C: sensitivity w.r.t. the states
- D: sensitivity w.r.t. the inputs
- u: set of inputs
- t: the evaluation time

Function 9

```
int getParamVarId(      char *paramname,
                        char *submodelname,
                        char *suffix);
```

Description

This is a function to get the ID values necessary to read the parameters and variables.

Inputs/outputs

- paramname: name of the parameter
- submodelname: name of the submodel
- suffix: "_.param" for parameters and "_.var" for variables

Function 10

```
void getStates(         char **allstates,
                        char *submodelname);
```

Description

This function extracts the states based on reading the AMESim files, the function provides a pointer to all the state names.

Inputs/outputs

- allstates: pointer to all the state names
- submodelname: name of the submodel

3. Step by step approach to use the functionalities

The user has to walk through several steps to perform a state estimation exercise, all of them can be tested by using the example:

1. Generate an AMESim model of the system
2. Use customized compiler to generate the model interfacing methods
3. Fill in the specific information into the ‘create header’ file
4. Compile & run the ‘create header’ executable to generate the header files
5. Compile & run the ‘MPCtool’ executable to run the simulation
6. The results of the different simulation steps are written in the AMESim results file.

The following functions have to be performed first and in the order as listed:

1. initialiseModelEvalfFunctions(model_name);
2. initialiseModelEvalfParameters(Ns,NoP,NoI,Obs,NofISteps,model_name,observations,input_vector,eps);
3. setTheSolver(dt,5);

In a follow-up step in the code, the model can be integrated and the sensitivities can be requested.